

Programação Estruturada

Entrada e Saída. Mais sobre dados

Professores Emílio Francesquini e Carla Negri Lintzmayer

2018.Q3

Centro de Matemática, Computação e Cognição
Universidade Federal do ABC



Saída de dados: printf

Para imprimir um texto, utilizamos o comando `printf`.

O texto deve ser uma constante do tipo **string**.

```
1 printf("Olá Pessoal!");
```

Saída:

```
1 Olá Pessoal!
```

No meio da constante **string** pode-se incluir caracteres de formatação especiais.

O símbolo especial `\n` é responsável por pular uma linha na saída.

```
1 printf("Olá Pessoal!\nOlá Pessoal");
```

Saída:

```
1 Olá Pessoal!  
2 Olá Pessoal
```

Escrevendo o conteúdo de uma variável na tela

Podemos imprimir também o conteúdo de variáveis.

Para isso utilizamos símbolos especiais no texto, para representar que aquele trecho deve ser substituído por uma variável ou constante.

No final do `printf` passamos uma lista de variáveis ou constantes, separadas por vírgula.

```
1 int a = 10;
2 printf("A variável %s contém o valor %d", "a", a);
```

Saída:

```
1 A variável a contém o valor 10
```

Escrevendo o conteúdo de uma variável na tela

No comando

```
1 int a = 10;
2 printf("A variável %s contém o valor %d", "a", a);
```

%s vai ser substituído por uma variável/constante do tipo `string` e %d vai ser substituído por uma variável/constante do tipo `int`.

É importante que a ordem na lista final de variáveis ou constantes seja a mesma ordem que os símbolos aparecem no texto.

O comando a seguir está errado:

```
1 int a = 10;
2 printf("A variável %s contém o valor %d", a, "a");
```

Formatos inteiros

%d — Escreve um inteiro na tela.

```
1 printf("%d\n", 10);
```

Saída:

```
1 10
```

```
1 int a = 12;  
2 printf("O valor é %d\n", a);
```

Saída:

```
1 O valor é 12
```

Formatos inteiros

O formato `%d` pode ser substituído por `%u` e `%ld` se desejarmos escrever variáveis `unsigned` ou `long int`, respectivamente.

```
1 printf("%d\n", 4000000000);
```

Saída:

```
1 -294967296
```

```
1 printf("%ld\n", 4000000000);
```

Saída:

```
1 4000000000
```

Formatos ponto flutuante

%f — Escreve um ponto flutuante na tela.

```
1 printf("%f\n", 10.0);
```

Saída:

```
1 10.000000
```

Formatos ponto flutuante

%.Nf — Escreve um ponto flutuante na tela com *N* casas decimais.

```
1 printf("%.2f\n", 10.1111);
```

Saída:

```
1 10.11
```

Formatos ponto flutuante

O formato `%f` pode ser substituído por `%lf` para escrever um `double` ao invés de um `float`.

```
1 double a = 10.0;  
2 printf("%.2lf\n", a);
```

Saída:

```
1 10.00
```

%c — Escreve um caractere.

```
1 printf("%c\n", 'A');
```

Saída:

```
1 A
```

Veremos que o comando `printf("%c\n", 65)` também imprime a letra A.

%s — Escreve uma **string**.

```
1 printf ("%s\n", "Meu primeiro programa");
```

Saída:

```
1 Meu primeiro programa
```

Entrada de datos: scanf

- Realiza a leitura de dados a partir do teclado.
- Parâmetros:
 - Uma **string**, indicando os tipos das variáveis que serão lidas e o formato dessa leitura.
 - Uma lista de variáveis.
- Aguarda que o usuário digite um valor e atribui o valor digitado à variável.

A função scanf

O programa abaixo é composto de quatro passos:

1. Cria uma variável **n**
2. Escreve na tela a string "Digite um número:"
3. Lê o valor do número digitado e o salva em **n**
4. Imprime o valor do número digitado

```
1  #include <stdio.h>
2  int main() {
3      int n;
4      printf("Digite um número: ");
5      scanf("%d", &n); /* note a presença do caractere & */
6      printf("O valor digitado foi %d\n",n);
7      return 0;
8  }
```

Formatos de leitura de variável

No `scanf`, cada variável para onde será lido um valor deve ser precedida do caractere `&`.

Os formatos de leitura são muito semelhantes aos formatos de escrita do `printf`.

Código	Função
<code>%c</code>	Lê um único caractere
<code>%s</code>	Lê uma série de caracteres (string)
<code>%d</code>	Lê um número decimal
<code>%u</code>	Lê um decimal sem sinal
<code>%ld</code>	Lê um inteiro longo
<code>%f</code>	Lê um número em ponto flutuante
<code>%lf</code>	Lê um double

A função scanf

```
1  #include <stdio.h>
2
3  int main() {
4      char c;
5      float b;
6      int a;
7
8      printf("Digite um caractere: ");
9      scanf("%c", &c);
10     printf("Digite um ponto flutuante: ");
11     scanf("%f", &b);
12     printf("Digite um inteiro: ");
13     scanf("%d", &a);
14
15     printf("Os dados lidos foram: %c, %f e %d.\n",c,b,a);
16
17     return 0;
18 }
```

A função scanf

No exemplo anterior, os textos “Digite ...” aparecem na saída padrão.

```
1  #include <stdio.h>
2
3  int main() {
4      char c;
5      float b;
6      int a;
7
8      scanf("%c", &c);
9      scanf("%f", &b);
10     scanf("%d", &a);
11
12     printf("Os dados lidos foram: %c, %f e %d.\n",c,b,a);
13
14     return 0;
15 }
```

A função scanf

É possível ler várias variáveis em um mesmo comando `scanf`, basta especificar todos os formatos das variáveis a serem lidas e depois as variáveis separadas por vírgulas.

```
1  #include <stdio.h>
2
3  int main() {
4      int m, n, o;
5      printf("Digite três números: ");
6      scanf("%d %d %d", &m, &n, &o);
7      printf("O valores digitados foram %d, %d e %d\n", m,
8          ↪ n, o);
9      return 0;
10 }
```

Informações extras sobre tipos

Possíveis formas de escrita em C de um número inteiro:

- Normalmente
Ex: 10, 145, 1000000
- Na forma hexadecimal (base 16), precedido de 0x
Ex: 0xA ($0xA_{16} = 10$), 0x100 ($0x100_{16} = 256$)
- Na forma octal (base 8), precedido de 0
Ex: 010 ($0x10_8 = 8$)

Constantes do tipo ponto flutuante

- Um número só pode ser considerado ponto flutuante se tiver uma parte “não inteira”, mesmo que ela tenha valor zero.
Ex: 10.0, 5.0, 4.7
- Um número ponto flutuante pode ser escrito em notação científica: um número seguido da letra **e** e um expoente. Um número escrito dessa forma deve ser interpretado como:

$$\text{numero} \times 10^{\text{expoente}}$$

Ex: 2e2 ($2 \cdot 10^2 = 200.0$)

- Caracteres são, na verdade, variáveis inteiras que armazenam um número associado ao símbolo
- Talvez a tabela de símbolos mais famosa utilizada pelos computadores é a tabela ASCII¹ (*American Standard Code for Information Interchange*), mas existem outras (EBCDIC, Unicode, etc.)
- Uma variável do tipo `char` armazena um símbolo (no caso, o inteiro correspondente ao símbolo). Seu valor pode ir de -128 a 127 (1 byte).

¹Pronúncia correta: ás-ki.

Tabela ASCII

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0 16	caracteres de Controle															
32		!	"	#	\$	%	&	'	()	*	+	,	-	.	/
48	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
64	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
80	P	Q	R	S	T	U	V	W	X	Y	Z	[\]	^	_
96	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
112	p	q	r	s	t	u	v	w	x	y	z	{		}	~	

Caracteres

Toda constante do tipo caractere pode ser usada como uma constante do tipo inteiro e vice-versa.

Nesse caso, o valor atribuído será o valor daquela letra na tabela ASCII.

```
1 int a;  
2 char b;  
3  
4 b = 65;  
5 printf("Valor de b: %c\n", b);  
6  
7 a = 'C';  
8 printf("Valor de a: %d\n", a);
```

Qual a saída do programa a seguir?

```
1 char b = 130;  
2  
3 printf("%c %d\n", b, b);
```

Comentários

- É comum incluir comentários em um programa para explicar certas partes do código e ajudar na sua compreensão.
- Os comentários são ignorados pelo compilador.
- Um comentário de uma linha deve ser escrito depois de `//`².
- Um comentário também pode ser escrito entre `/*` e `*/`. Neste caso pode haver mais de uma linha de comentário.

²Comentários de uma linha não são permitidos em ANSI C.

```
1  #include <stdio.h>
2
3  int main() {
4      int a; //isto é um comentário
5
6      int b; /* isto
7              também é
8              um comentário */
9
10     a = 2;
11     b = 3;
12     a = a + b;
13
14     return 0;
15 }
```

Conversão de tipos

- É possível converter alguns tipos entre si, o que pode ser da forma implícita ou explícita.
- Na forma **implícita**, atribui-se diretamente um dado de um tipo para uma variável de outro tipo.
- Para isso, a capacidade (tamanho) do destino deve ser maior ou igual ao tamanho da origem, senão há perda de informação.

Conversão implícita de tipos

```
1  #include <stdio.h>
2  int main() {
3      int a = 9;
4      double b;
5
6      b = a;
7      printf("a: %d e b: %lf\n", a, b);
8
9      b = 5.56;
10     a = b;
11     printf("a: %d e b: %lf\n", a, b);
12
13     b = 4000000000.56;
14     a = b;
15     printf("a: %d e b: %lf\n", a, b);
16
17     printf("Tamanho em bytes de um double: %ld\n", sizeof(double));
18     printf("Tamanho em bytes de um int: %ld\n", sizeof(int));
19     return 0;
20 }
```

Conversão de tipos

Na conversão **explícita**, nós explicitamente informamos o tipo para o qual o valor da variável ou expressão será convertido.

```
1  #include <stdio.h>
2
3  int main() {
4      double b;
5
6      b = ((double) 5 / (double) 2);
7      printf("%lf\n", b);
8
9      b = 5 / 2;
10     printf("%lf\n", b);
11
12     return 0;
13 }
```

Um uso da conversão de tipos

A operação de divisão (/) possui dois modos de operação de acordo com os seus argumentos: inteira ou de ponto flutuante.

- Se os dois argumentos forem inteiros, acontece a divisão inteira. A expressão $10 / 3$ tem como valor 3.
- Se **um** dos dois argumentos for de ponto flutuante, acontece a divisão de ponto flutuante. A expressão $1.5 / 3$ tem como valor 0.5.

Quando se deseja obter o valor de ponto flutuante de uma divisão (não-exata) de dois inteiros, basta converter um deles para ponto flutuante: a expressão $10 / (\text{float}) 3$ tem como valor 3.33333333.

Isso é especialmente útil quando se tem variáveis na expressão.